MOOC Intro POO C++
Exercices semaine 6

Exercice 20 : animaux en peluche (niveau 1)

> Cet exercice correspond à l'exercice n°63 (pages 160 et 351) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Une société protectrice des animaux souhaite commercialiser des animaux en peluche à l'effigie de ses protégés pour renflouer ses caisses.

Elle possède déjà un programme C++ où les animaux sont représentés par une classe, `Animal`, dotées des attributs et méthodes suivantes:

- un attribut `nom` (chaîne de caractère)
- une attribut `continent` (chaîne de caractère) indiquant dans quel continent l'animal vit (on supposera pour simplifier qu'il n'y a qu'un continent par animal).
- une méthode `affiche` affichant une message du type « `Je suis un nom et je vis en continent.` » (Exemple: « `Je suis un Panda et je vis en Asie.` »)

Les animaux en danger y sont représentés par une autre classe, `EnDanger`, dotée:

- d'un attribut `nombre` (entier) indiquant le nombre d'individus recensés sur terre
- d'une méthode `affiche` affichant une message du type « `Il ne reste que nombre de mon espèce sur Terre !` » (Exemple : « `Il ne reste que 200 individus de mon espèce sur Terre !` »)

Une troisième classe, `Gadget`, existe dans ce programme. Elle représente l'ensemble des "gadgets" commercialisés par notre société et contient les attributs et méthodes suivants :

- un attribut `nom` (chaîne de caractère) indiquant le nom du produit
- un attribut `prix` (nombre réel) indiquant le prix de l'objet
- une méthode `affiche` affichant une message du type « `Mon nom est nom` » (Exemple : « `Mon nom est Ming.` »)
- une méthode `affiche_prix` affichant une message du type « `Achetez-moi pour prix francs et vous contribuerez à me sauver !` » (Exemple : « `Achetez-moi pour 20 francs et vous contribuerez à me sauver !)` »

Dans un fichier `peluches.cc`, reproduisez le codage de ces classes.

Dotez ces classes de constructeurs prenant l'ensemble des attributs nécessaires comme paramètres et affichant les messages:

- « `Nouvel animal protégé` » pour la classe `Animal`
- « `Nouveau gadget` » pour la classe `Gadget`
- « `Nouvel animal en danger` » pour la classe `EnDanger`

Dotez également ces classes de destructeurs affichant les messages:

- « `Je ne suis plus protégé` » pour la classe `Animal`
- « `Je ne suis plus un gadget` » pour la classe `Gadget`
- « `ouf! je ne suis plus en danger` » pour la classe `EnDanger`

Animaux en peluches

Notre société souhaite doter ses peluches d'étiquettes décrivant les caractéristiques de l'animal représenté.

Définissez une classe `Peluche` héritant des classes `Animal`, `EnDanger` et `Gadget` (eh oui.. C++ permet quelques bizarreries génétiques !).

Dotez votre classe `Peluche` d'une méthode `etiquette` affichant le texte à mettre sur l'étiquette de la peluche.

Ce texte aura la forme suivante:

```
Hello,
Mon nom est Ming.
Je suis un Panda  et je vis en Asie.
Il ne reste que 200 de mes congénères sur terre.
Achetez moi pour 20 francs et vous contribuerez
à me sauver!
```

la méthode `etiquette` devra être codée au moyen des méthodes `affiche` et `affiche_prix` des super-classes.

Dotez également votre classe d'un constructeur et d'un destructeurs affichant des messages analogues à ceux des super-classes.

Testez votre programme au moyen du main suivant et observez l'ordre d'invocation des constructeurs/destructeurs :

```
int main()
{
  Peluche panda("Panda","Ming","Asie", 200, 20.0);
  Peluche serpent("Cobra","Ssss","Asie", 500, 10.0);
  Peluche toucan("Toucan","Bello","Amérique", 1000, 15.0);


  panda.etiquette();
  serpent.etiquette();
  toucan.etiquette();

  return 0;
}
```

Exercice 21 : employés (niveau 2)

> Cet exercice correspond à l'exercice n°64 (pages 162 et 354) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Cet exercice vous permettra de concevoir une hiérarchie de classes comportant de l'héritage multiple. Il vous servira également de révision pour la notion de collections hétérogène vue la semaine passée.

Le directeur d'une entreprise de produits chimiques souhaite gérer les salaires et primes de ses employés au moyen d'un programme C++.

Un employé est caractérisé par son nom, son prénom, son âge et sa date d'entrée en service dans l'entreprise.

Dans un fichier `salaires.cc`, codez une classe abstraite `Employe` dotée des attributs nécessaires, d'une méthode virtuelle pure `calculer_salaire` (ce calcul dépendra en effet du type de l'employé) et d'une méthode virtuelle `get_nom` retournant la chaîne de caractères `"L'employé "` suivit du prénom et du nom.

Dotez également votre classe d'un constructeur prenant en paramètre l'ensemble des attributs nécessaires et d'un destructeur virtuel vide.

Calcul du salaire

Le calcul du salaire mensuel dépend du type de l'employé. On distingue les types d'employés suivants :

- Ceux affectés à la *Vente*. Leur salaire mensuel est le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 400 Francs.
- Ceux affectés à la *Représentation*. Leur salaire mensuel est également le 20 % du *chiffre d'affaire* qu'ils réalisent mensuellement, plus 800 Francs.
- Ceux affectés à la *Production*. Leur salaire vaut le *nombre d'unités* produites mensuellement multipliées par 5.
- Ceux affectés à la *Manutention*. Leur salaire vaut leur *nombre d'heures* de travail mensuel multipliées par 65 francs.

Codez dans votre fichier `salaires.cc` une hiérarchie de classes pour les employés en respectant les conditions suivantes :

- La super-classe de la hiérarchie doit être la classe `Employe`.
- Les nouvelles classes doivent contenir les attributs qui leur sont spécifiques ainsi que le codage approprié des méthodes `calculer_salaire` et `get_nom` (en changeant le mot "employé" par la catégorie correspondante).
- Chaque sous classe est dotée de constructeur prenant en argument l'ensemble des attributs nécessaires et d'un destructeur.
- N'hésitez pas à introduire des classes intermédiaires pour éviter au maximum les redondances d'attributs et de méthodes dans les sous-classes

Employés à risques

Certains employés des secteurs *production* et *manutention* sont appelés à fabriquer et manipuler des produits dangereux.

Après plusieurs négociation syndicales, ces derniers parviennent à obtenir une prime de risque mensuelle.

Complétez votre programme `salaires.cc` en introduisant deux nouvelles sous-classes d'employés. Ces sous-classes désigneront les employés des secteurs *production* et *manutention* travaillant avec des produits dangereux.

Ajouter également à votre programme une nouvelle **super-**classe pour les *employés à risque* permettant de leur associer un attribut *prime mensuelle*. Cette classe sera dotée d'un constructeur initialisant la prime mensuelle à 100 francs par défaut. Elle fournira également un destructeur virtuel vide.

Collection d'employés

Satisfait de la hiérarchie proposée, notre directeur souhaite maintenant l'exploiter pour afficher le salaire de tous ses employés ainsi que le salaire moyen.

Ajoutez une classe `Personnel` contenant une « collection » d'employés. Il s'agira d'une collection « hétérogène »

d'`Employe`.
Vous pouvez aussi choisir de faire hériter cette classe de la classe « tableau dynamique de pointeurs sur des `Employe` », et alors utiliser la méthode `push_back`.

Prototypez et définissez ensuite les méthodes suivantes à la classe `Personnel` :

- `void ajouter_employe(Employe*)`

  qui ajoute un (pointeur sur) un employé à la collection.

- `void calculer_salaires() const`

  qui affiche le salaire de chacun des employés de la collection.

- `double salaire_moyen() const`

  qui affiche le salaire moyen des employés de la collection.

Dotez également votre classe `Personnel` d'une méthode `licencie` permettant de supprimer tous les employés stockés dans la collection, tout en libérant leur espace mémoire.

Testez votre programme avec le main suivant :

```
int main () {
  Personnel p;
  p.ajouter_employe(new Vendeur("Pierre", "Business", 45, "1995", 30000));
  p.ajouter_employe(new Representant("Léon", "Vendtout", 25, "2001", 20000));
  p.ajouter_employe(new Technicien("Yves", "Bosseur", 28, "1998", 1000));
  p.ajouter_employe(new Manutentionnaire("Jeanne", "Stocketout", 32, "1998", 45));
  p.ajouter_employe(new TechnARisque("Jean", "Flippe", 28, "2000", 1000, 200));
  p.ajouter_employe(new ManutARisque("Al", "Abordage", 30, "2001", 45, 120));

  p.afficher_salaires();
  cout << "Le salaire moyen dans l'entreprise est de "
       << p.salaire_moyen() << " francs." << endl;

  // libération mémoire
  p.licencie();
}
```

Vous devriez obtenir quelque chose comme :

```
Le vendeur Pierre Business gagne 6400 francs.
Le représentant Léon Vendtout gagne 4800 francs.
Le technicien Yves Bosseur gagne 5000 francs.
Le manut. Jeanne Stocketout gagne 2925 francs.
Le technicien Jean Flippe gagne 5200 francs.
Le manut. Al Abordage gagne 3045 francs.
Le salaire moyen dans l'entreprise est de 4561.67 francs.
```

Exercice 22 : jeu de cartes (niveau 2)

Cet exercice correspond à l'exercice n°65 (pages 164 et 358) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Un jeune programmeur veut tenter d'appliquer ses nouvelles connaissance en C++ à la gestion d'un de ses hobbies, un jeu de cartes simulant des combats de magiciens.

Dans ce jeu, il existe trois types de cartes : les « terrains », les « créatures » et les « sortilèges ».

Les terrain possèdent une couleur parmi cinq : blanc, bleu, noir, rouge et vert.

Les créatures possèdent un nom, un nombre de points de dégâts et un nombre de points de vie.

Les sortilèges possèdent un nom et une explication (sous forme de texte).

De plus, chaque carte, indépendamment de son type, possède un coût. Le coût d'un terrain est nul.
Attention au piège : `cout` est déjà un mot réservé du langage (lorsqu'on utilise l'espace de noms `std`) !!

Dans un programme `magic.cc`, proposer (et implémenter) une hiérarchie de classes permettant de représenter des cartes de différents types.

Chaque classe aura un constructeur permettant de spécifier les valeurs de ses attributs. De plus, chaque constructeur devra afficher le type de la carte.

Le programme doit utiliser la conception orientée objets et ne doit pas comporter de duplication de code.

Ajouter ensuite aux cartes une méthode `afficher` qui, pour toute carte, affiche son coût et la valeur de ses arguments spécifiques.

Créer de plus une classe pour représenter un jeu de cartes, c.-à-d. une collection de telles cartes.

Cette classe devra avoir une méthode `ajouter` permettant d'ajouter une carte au jeu (de sorte à pouvoir utiliser le polymorphisme sur les cartes par la suite). Il sera peut-être utile d'ajouter une méthode libérant le contenu du jeu de carte.

Il existe de plus certaines cartes particulières qui peuvent être à la fois un terrain et une créature. Proposer une nouvelle classe pour de telles cartes. La méthode `afficher` devra dans ce cas donner toutes les informations relatives à la carte (i.e. sa couleur et le nombre de points de dégâts et de points de vie).

Pour finir, constituer dans le `main()` un jeu contenant divers types de cartes et faites afficher le jeu grâce à une méthode `afficher` propre à cette classe.

Par exemple, le `main` pourrait ressembler à (nécessite peut-être des adaptations) :

```
  Jeu mamain;

  mamain.ajoute(new Terrain(BLEU));
  mamain.ajoute(new Creature(6, "Golem", 4, 6));
  mamain.ajoute(new Sortilege(1, "Croissance Gigantesque",
              "La créature ciblée gagne +3/+3 jusqu'à la fin du tour"));
  mamain.ajoute(new CreatureTerrain(2, "Ondine", 1, 1, BLEU));

  mamain.afficher();
```

et produirait un résultat ressemblant à :

```
On change de main
Un nouveau terrain.
Une nouvelle créature.
Un sortilège de plus.
Une nouvelle créature.
Un nouveau terrain.
Houla, une créature/terrain.
```

```
Là, j'ai en stock :
  + Un terrain bleu.
  + Une créature Golem 4/6 de coût 6
  + Un sortilège Croissance Gigantesque de coût 1
  + Une créature/terrain bleue Ondine 1/1 de coût 2
Je jette ma main.
```

NOTE : Les méthodes `afficher` peuvent bien sûr être remplacées par l'opérateur <<.
